ToothPic: camera-based image retrieval on large scales

ToothPic: camera-based image retrieval on large scales

Diego Valsesia, Giulio Coluccia, Tiziano Bianchi, Enrico Magli

In the early years 2000s, digital cameras appeared on the market for non-professional photographers. Soon, their price dropped, and more and more people could afford buying one. Between 2006 and 2007, the most important photo-centric social network, Facebook, was opened to the masses giving them the chance to share their pictures with the entire world. Facebook user count increased from 100 Million people in 2008 to 2 Billion in 2017. In the following years, the advent of camera-equipped smartphones, the availability of an ubiquitous Internet connection, and the multiplicity of diverse photo-sharing platforms made the number of photos shared online explode to 2.5 Trillion.

Of course, the vast majority of those pictures portray legal content and are shared by their owner in a legitimate way. However, in some cases there may be misuses or abuses related to the content of a picture, or the way this picture is shared. For example, a picture taken by a photographer may be stolen from him/her and used without his/her consent for commercial purposes. Or even worse, a picture may contain illegal content, e.g. child pornography or an act of bullying, and may rapidly spread hopping from a chat group to another one. As a final example, the shady world of the deep web presents every kind of illegal material like photos related to international terrorism. In all these cases, being able to reliably link a picture to the device that shot it is of paramount importance to give credit or assign responsibility to the author of the picture itself. However, this task needs to be performed at the large scales dictated by the examples above.

Existing methods cannot satisfy those requirements. Photo-related metadata, like the EXIF data, can be easily tampered with and they are often automatically stripped from the pictures by the sharing platforms. Other methods, based on the Photo Response Non-Uniformity (PRNU) of digital sensors are able to link a photo to the device that shot it (see box *Camera identification with PRNU fingerprints*) and have already been used as proof in the Court of Law. Those methods are reliable but so far they can be only used for small-scale forensic tasks involving few cameras and pictures.

A. ToothPic Image Retrieval

ToothPic, an acronym for *Who Took This Picture?*, is an image retrieval engine that allows to create a database of pictures, collected from any source and possibly growing over time, and a database of cameras, built by having access to some high-quality photos known to be from a specific device. Queries can be performed to retrieve all the pictures in the database shot by a given query camera. ToothPic uses the PRNU pattern of digital camera sensors, which has been proven to be a very discriminative fingerprint of the device [1]. The PRNU is compressed with a random projection method [2] to achieve very compact representations that are also invariant to image rescaling and enable fast search [3], allowing the system to work on large scales. In particular, ToothPic is robust to:

- light post-processing (e.g. color, contrast enhancement, etc.);
- image rescaling preserving the aspect ratio;
- most common automatic image crops (e.g., 4:3 to 16:9 vertical/horizontal crops, square crops, ...). Those crops correspond to operations performed by the most popular social networks and messaging applications (see box *Robustness to automatic image crop and rescaling* for details). For example, the photo acquisition in Instagram automatically crops the central square area of the sensor, while WhatsApp automatically crops pictures in 16:9 format when shot from a smartphone with a 4:3 sensor;
- image rotations at multiples of 90 degrees.

I. System Architecture

An overview of the architecture is shown in Fig.1. The following sections describe the main functional blocks including the camera registration and photo insertion procedures, the compressed format of PRNU fingerprints, and the two-stage search performed when the system is queried with a registered camera in order to obtain all the photos shot by that device.

ToohPic is built upon some key techniques allowing effective representations of the PRNU sensor fingerprints: binary-quantized random projections, two-stage search with adaptive embeddings, scale-invariant compressed fingerprints and crop-robust camera registration. We will now discuss how such techniques are used in the overall system architecture.

A. Scale-invariant fingerprint compression

Uncompressed PRNU fingerprints have huge size since they have as many real-valued coefficients as the number of pixels in the camera sensor (see box *Camera identification with PRNU fingerprints*). Because of their noise-like nature, they are not amenable to standard image compression techniques like JPEG. In [2] we showed that random projections can be used to reduce the PRNU fingerprint, sizing tens of megabytes, by three orders of magnitude without significant loss in detection and false alarm rates. In ToothPic, we use an improved version of the compression technique which allows to generate compressed fingerprints that are invariant

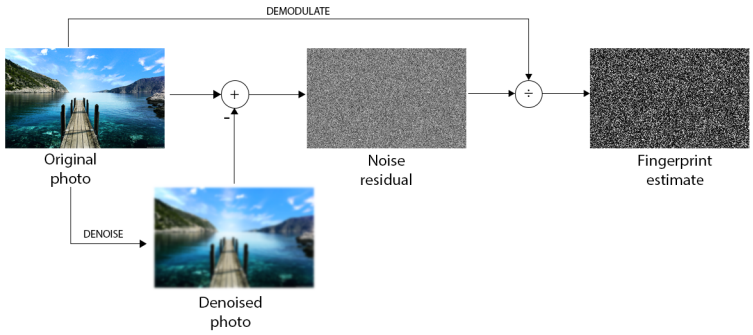
Camera identification with PRNU fingerprints

Camera identification is concerned with linking a picture with the device that shot it in a reliable and robust manner. An established technique is to use the Photo Response Non-Uniformity (PRNU) pattern of digital imaging sensors as a unique fingerprint of the device. Digital imaging sensors use matrices of photo detectors (pixels) to convert photons into electrons by means of the photoelectric effect. However, not all pixels respond to light in the same manner due varying quantum efficiency caused by inhomogeneities in the silicon wafer or uncertainties in the physical area of the detector. This effect can be modeled as a multiplicative pattern superimposed on the intensity of every picture that is taken:

$$\mathbf{O} = \mathbf{O}^{\text{id}} + \mathbf{O}^{\text{id}} \cdot \mathbf{K} + \mathbf{E} \ , \tag{1}$$

being $\mathbf{O}$ the sensor output, $\mathbf{O}^{\text{id}}$ the actual light intensity and $\mathbf{K}$ the PRNU pattern pixelwise multiplied to $\mathbf{O}^{\text{id}}$. Being a multiplicative factor, the PRNU is only weakly observed in dark scenes and it is lost in saturated areas of the image where the intensity exceeds the dynamic range.

The PRNU pattern can be estimated by one or multiple photos by obtaining an approximation of $\mathbf{O}^{\text{id}}$ with a denoising filter, subtracting it from $\mathbf{O}$ and demodulating the resulting "noise residual" again by such approximation. Some post-processing operations are also typically applied as some artifacts due to the image processing pipeline are present which may be shared among cameras of the same brand or model [4].



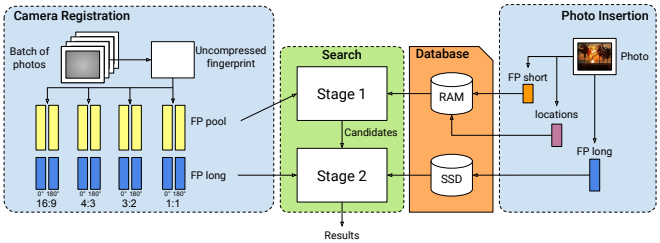


Fig. 1. System architecture. When a new photo is inserted in the system (*Photo Insertion*), a very compact representation of the fingerprint (FP short+ locations) is generated and stored in RAM database, to be used during the *Stage 1* of the search process, and a larger representation (FP long) is also computed and stored in SSD database, to be used during the *Stage 2*. When registering a camera (*Camera Registration*) the compressed fingerprint is computed for 4 aspect ratios, corresponding to different sensor crop areas, and 2 reference rotations. For each of them the representations needed by the *Stage 1* (FP pool) and *Stage 2* (FP long) are stored.

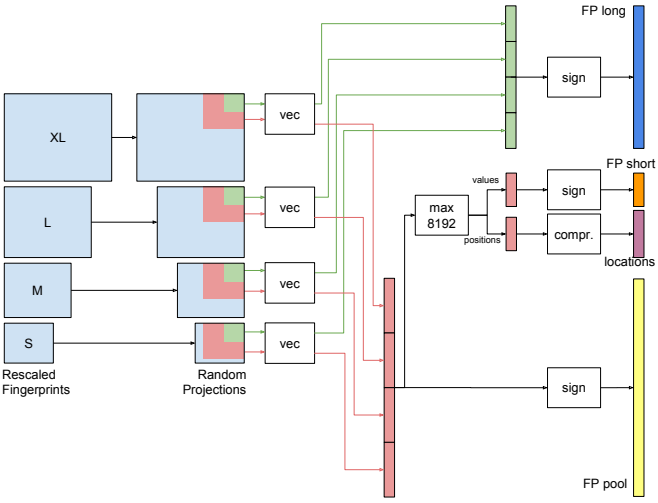


Fig. 2. Scale-invariant compression procedure.

to scale transformations of the source images. The procedure is depicted in Fig. 2.

First, the uncompressed fingerprint is computed using standard techniques in the camera identification literature and kept in landscape orientation. Then, the fingerprint is rescaled to four standard scales corresponding to fixed horizontal resolutions reported in Table I. Such resolutions are chosen among the most commonly used for each of the supported aspect ratios. For each resolution, the rescaled fingerprint is circularly convolved in two dimensions with a fixed random pattern of the same size. Such pattern is composed of realizations of independent and identically distributed standard Gaussian random variables and implements the random projection functionality. Notice that this operation can be efficiently performed using the 2D Fast Fourier Transform:

$$\mathbf{Y}_i = \text{IFFT}\left[\text{FFT}\left[\boldsymbol{\phi}_i\right] \cdot \text{FFT}\left[\mathbf{X}_i\right]\right] \tag{2}$$

TABLE I
HORIZONTAL RESOLUTIONS FOR EACH ASPECT RATIO

AR	S	M	L	XL
3:2	960	1600	2048	5184
4:3	960	1600	2048	3264
16:9	1920	2048	3072	3584
1:1	1080	1280	1600	2048

being $\mathbf{X}_i$ the uncompressed fingerprint rescaled at resolution i, $\boldsymbol{\phi}_i$ the random pattern at resolution i and $\mathbf{Y}_i$ the corresponding random projections.

Each matrix $\mathbf{Y}_i$ is then subsampled by keeping only the upper right corner of size 320×410 pixels to obtain $\tilde{\mathbf{Y}}_i$. The resulting matrices are then vectorized into $\tilde{\mathbf{y}}_i$, stacked and binary-quantized to obtain the final compressed fingerprint, that we call FP long fingerprint:

$$\mathbf{y} = \begin{bmatrix} \text{sign}(\tilde{\mathbf{y}}_{\mathsf{S}}) \\ \text{sign}(\tilde{\mathbf{y}}_{\mathsf{M}}) \\ \text{sign}(\tilde{\mathbf{y}}_{\mathsf{L}}) \\ \text{sign}(\tilde{\mathbf{y}}_{\mathsf{XL}}) \end{bmatrix} . \tag{3}$$

The FP long fingerprint is exactly 65600 bytes long and it will be stored on disk.

The fingerprint compression procedure also creates the so-called FP short compressed fingerprint which is smaller and will be stored in RAM in order to have a fast first stage during searches. The FP short compressed fingerprint is an adaptive embedding obtained by selecting from a pool of random projections the ones with largest magnitude before quantization. A theoretical analysis of the properties of such embedding is available in [5]. The procedure to compute FP short starts from the random projection matrices $\mathbf{Y}_i$. Each $\mathbf{Y}_i$ is subsampled to a size 630×820. This crop forms the pool of random projections among which the ones with largest magnitude are selected. When registering a camera, these pools are binary-quantized and separately saved as the FP pools to be used at query time. Instead, when a photo is being inserted, we vectorize and stack all the pools, as in (3), and select the 8192 entries with largest magnitude, binary-quantize their values and store in the locations their position in the vector of pools. The quantized values form the FP short compressed fingerprint, which is therefore 1024 bytes long, while the locations are extra information that is also stored in RAM and used to select the right bits from the FP pool fingerprint during the first stage of the search. Notice that the locations are stored in a compressed format [3]. This makes the locations of variable size. However, we experimentally verified that the typical size is around 11 KiB.

B. Crop-robust camera registration

The fingerprint compression procedure solved the issue of the robustness to scale transformations. However, many common usage scenarios involve photos that have been cropped with respect to the original sensor size. While managing arbitrary crops and scale transformations at the same time is a hard open problem, handling standard non-arbitrary crops already covers several use cases. The most evident example is the automatic crop performed to pictures by social networks or messaging applications. For example, it is common to crop images to fit the aspect ratio of smartphone displays: e.g. a 4:3 sensor to be cropped to fill a 16:9 screen. Such kind of crop is predictable as it is applied in fixed positions (typically full-height centered). The camera registration phase in the ToothPic system prepares multiple versions of the compressed sensor fingerprint by applying the most common standard crops (16:9 to 4:3 and vice-versa and 16:9/4:3 to 1:1). When a camera is used as a query, multiple queries are actually performed to test all the crop factors.

C. Rotation robustness

Robustness to rotations of multiples of 90 degrees is achieved by performing two queries, one where the camera fingerprint is in the reference orientation and one where it is rotated by 180 degrees. Notice that this covers all the 4 angle rotations since photos are always rotated to be in a landscape orientation during insertion, thus only having two states of rotational ambiguity (apart from the 1:1 aspect ratio, for which 4 rotations are stored).

D. Two-stage search

A query is performed by selecting a camera in order to retrieve all the photos shot by that device present in the picture database. The first operation performed by the first stage of the search is to filter the photos in the database by retaining only the ones matching the aspect ratio of the query. Then, for each of the retained entries, the locations are decompressed and used to select the corresponding entries in the FP pool compressed fingerprint used as query. Once the binary values corresponding to the locations have been obtained, the Hamming distance with the FP short compressed fingerprint of the photo under test is computed. If the Hamming distance is below a predefined threshold, then the photo is selected for the second stage. As the computational complexity of the first stage scales linearly with the number of photos in the database, the FP short and locations of each photo are stored in RAM, in order to achieve maximum speed. The FP long compressed fingerprints of the photos that were selected by the first stage are loaded from disk and the Hamming distance with the FP long compressed fingerprint of the query is computed. If the Hamming distance is below a predefined threshold, then the photo is declared as being shot by the query camera. This two-stage procedure is applied for all the crop factors registered for the query camera and, for each crop factor, the two rotations are tested (4 in the case of 1:1 aspect ratio).

II. IMPLEMENTATION

The implementation of the system presented in the previous section strives to be computationally efficient and scalable to handle a large number of images. In particular, the core component affecting the system architecture and performance is the database system. The selection of a suitable database system hinged on the fulfillment of three main requirements:

- Hybrid memory architecture: the database should be able to operate in-memory, i.e. storing some of the data

directly in RAM, but should also support disk storage. Also, optimization for SSD drives is desirable since the second stage of the search procedure is a read intensive operation.
- Distributed architecture: the database should seamlessly scale to handle clusters with an arbitrary number of machines.
- Distributed data processing: queried data should be processed in a MapReduce-like [6] fashion with computations that are localized to the machine storing the data and results that are aggregated over the network.

In light of such requirements, the Aerospike database [7] was chosen. Aerospike is a NoSQL database [8] working as a key-value store and with a storage system that is optimized to use RAM, Flash and SSD memory. It also supports the creation of clusters to distribute data and processing operations. User-Defined Functions (UDFs) support writing application-dependent code that runs inside the Aerospike database server. UDFs are typically used to perform computations on the data records. In particular, we are interested in the Stream UDFs: they perform distributed stream processing, i.e., processing a stream of records returned from a query directly on each node in the cluster and aggregating the results in the client application. UDFs are written in LUA [9] and can use C code called from a LUA wrapper. We used this feature writing Stream UDFs that call C code to perform the matching operations described below.

In order to implement the ToothPic system with Aerospike, we first created two namespaces: the first one is used to hold data in RAM and has a single set of records, the second one is for SSD storage and has two sets of data. The data set in the RAM namespace has one record for each photo we inserted, and the following bins:

- unique identifier
- FP short
- locations
- aspect ratio

To avoid custom aspect ratios due to unsupported crops or merging of multiple photos, only photos matching one of the predefined aspect ratios are inserted. A secondary index is created on the aspect ratio bin in order to query all the records matching a chosen aspect ratio in a fast way.

The first data set in the SSD namespace matches the one in RAM, i.e., has one record for each photo we inserted, and the following bins:

- unique identifier (same as in the RAM data)
- FP long
- validity flag

The validity flag is a binary flag with a secondary index that is used to mark which photos have passed the first stage during search.

The second data set in SSD namespace has one record per camera with the path on the disk where the FP pool and FP long of the camera are stored for each query aspect ratio.

A query is initiated by selecting a camera from the cameras data set. The secondary index on the aspect ratio bin of the data set in the RAM namespace is queried to return a stream of records matching that aspect ratio. This stream is processed by a Stream UDF: a LUA wrapper to C code implements decompression of the locations, selection of the corresponding bits in the FP pool, Hamming distance computation and comparison with a user-supplied threshold. These operations run locally on every machine in the cluster and each machine produces a list of unique identifiers, i.e., the records that passed the first stage. The lists are then aggregated by the client application and the identifiers are used to set the validity flag to *valid* on the corresponding records in the SSD namespace. Notice that this solution is suboptimal since it requires disk write operations, albeit very limited in size, but we have found no alternative mechanism since it is not possible to initiate a second Stream UDF directly on the list of record identifiers. Instead, the Stream UDF is initiated against the secondary index corresponding the the validity flag bin. This second Stream UDF also calls some C code to compute the Hamming distance between the FP long fingerprint of the camera and of the photo record. This procedure is repeated for all registered crop factors and rotations.

III. Performance

We tested our implementation of the ToothPic image retrieval system on a large-scale scenario in order to assess its performance in terms of speed, precision and recall metrics. We also analyzed the robustness of the system to image rescaling and automatic processing introduced by common social networks and messaging applications, which perform automatic crop and/or rescaling operations.

For testing purposes we assembled a database of about 26 million publicly available images downloaded from Flickr. A list of anonymized Flickr users has been created and for each user, his/her publicly available photos have been downloaded. Since Flickr stores different resolutions of the same picture, only the highest available resolution has been considered. All the pictures fulfilling the following criteria have been automatically downloaded in an *unsupervised* way:

- Size: between 512,000 and 22,000,000 pixels
- Aspect Ratio: one among 16:9, 4:3, 3:2, 1:1

Those requirements have been chosen to automatically discard thumbnails, panoramic pictures, unconventional crops, etc. Portrait pictures have been rotated to landscape. EXIF data have been have not been used. For each downloaded picture, the compressed fingerprint has been created and inserted in the database according to the procedure described in the previous sections. In order to be able to measure the system performance in terms of accuracy in the retrieval task, we augmented the database of pictures from Flickr with the well-known Dresden Image database [10], which includes a set of natural and flatfield pictures, labelled with the camera that shot them. We included in our database the natural photos, while the flatfield photos of the Dresden database were used to generate the queries to the system. Table II reports the total number of photos in the database and a breakdown by aspect ratio: the Dresden database accounts for more than 10,000 natural pictures and 53 cameras.

The database has been installed and runs on a cluster with two servers, each employing two CPUs with 20 cores each,

TABLE II
TEST DATABASE

No. of photos	26,110,823
3:2	14,937,869
4:3	9,707,557
16:9	1,063,088
1:1	402,309

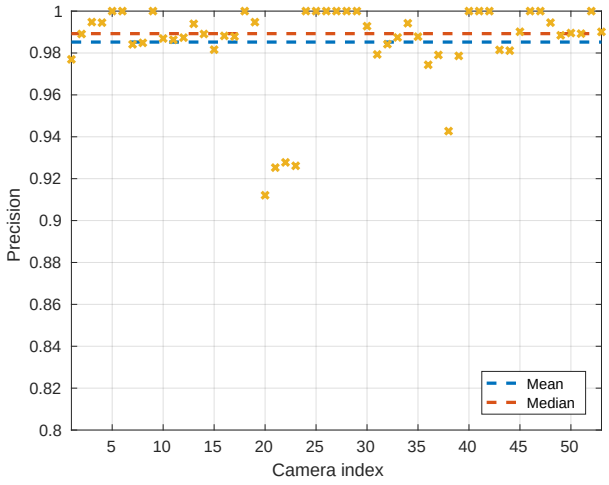


Fig. 3. ToothPic system precision per camera.

384 GB of RAM, 1.6TB of fast PCIe SSD storage. The client application used to query the database is written in Java using the Aerospike-Java interface and runs on one of the servers.

We queried the system with each of the 53 cameras of the Dresden database. For each query, we measured the following quantities:

- True (False) Positives (TP, FP): the number of retrieved photos that have (have not) actually been shot by the selected camera
- True (False) negatives (TN, FN): the number of unretrieved photos that have *not* (have) been shot by the selected camera

Hence, the system performance has been measured using the following metrics:

- Precision: TP/(TP+FP), i.e., the percentage of retrieved photos that have actually been shot by the selected camera;
- Recall: TP/(TP+FN), i.e., the percentage of the natural photos of the selected camera that have been retrieved by the search engine;
- Execution Time: time spent by the system to complete a query. It has been measured using the Linux `time` command when invoking the query script. We used a query script that sends a request to the Java client and waits until the results are ready.

A. Results

The system has been queried using the compressed fingerprints generated from the flatfield pictures of each of the 53 cameras of the Dresden dataset. The threshold in terms of Hamming distance for the first stage is equal to 3969 bits,

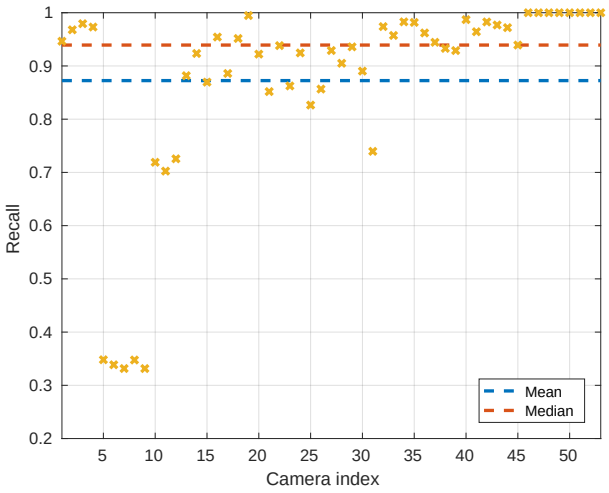


Fig. 4. ToothPic system recall per camera.

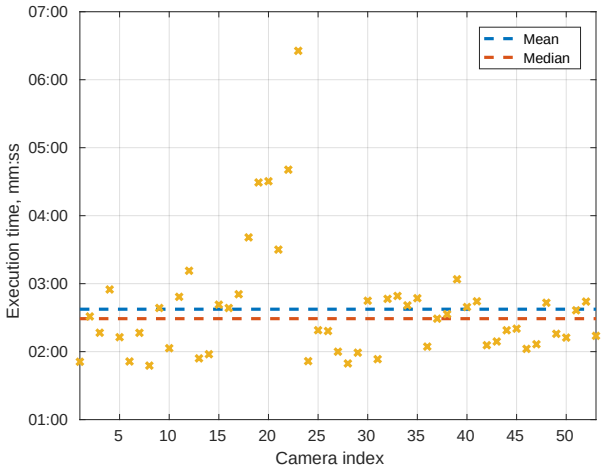


Fig. 5. ToothPic system execution time per camera.

tuned to a probability of about 10^{-3} of passing the first stage for a random picture. The default threshold for the second stage is equal to 260500, as an optimal compromise between precision and recall.

We first tested the system using the natural photos at their original resolution. The value of the three different metrics is shown in Fig. 3 (precision), Fig. 4 (recall), and Fig. 5 (execution time). Those figures also report the mean and median values of the metrics measured for the 53 tested cameras.

The system performance in terms of accuracy in the retrieval task and execution time is summarized in Table III.

The obtained results in terms of precision and recall agree with the results published in scientific literature, where the Dresden database was used as reference, see for example [3] and [11]. Given that the database contains 26,110,823 pictures, the median execution time corresponds to a throughput (in terms of processed images per second) equal to more than 175,000 pictures/second.

When evaluating the results, it has to be noted that:

- The Dresden database is known to contain a few cameras

Robustness to automatic image crop and rescaling

The rise of smartphone photography has increased the number of photos taken and shared by means of social networks or messaging applications every day. However, it is not common to find them at their original resolution due to a variety of reasons. Most pictures will be seen on mobile devices with screens of limited size and therefore rescaling the photo to a lower resolution does not significantly alter the visual quality, while allowing reduced upload and download latency, or reduced storage requirements for service providers. Sometimes images are also cropped from the original sensor form factor in order to match the screen size of a target device or for aesthetic reasons (e.g., square photos). We report the processing operations performed by some popular services and test how well the ToothPic system handles camera-based image retrieval in this use case of great practical interest. The tests use a Nexus 5X smartphone with original sensor size of 4092×3024 pixels.

WhatsApp

When sending a photo shot using the in-app camera, the image is automatically cropped to a 16:9 aspect ratio by keeping the central part of the image and resized. The sent images have resolution 899×1599. Notice that the app saves on the sender's phone both the cropped and resized photo and only the cropped photo, while the recipient only receives the former. We simulate scenarios where we could access the file system of the sender or of the recipient.

	SENDER	RECIPIENT
Recall	20/20	4/20
Precision	20/20	4/4

Facebook

Photo upload resizes the photos to one of the supported horizontal resolutions: 720px, 2048px ("high quality" option). No crop is performed.

	720px	2048px
Recall	9/20	16/20
Precision	9/9	16/16

Google Photos

The free tier ("high quality" option) allows to save unlimited photos but caps the resolution to 16 megapixels and uses stronger JPEG compression. No crop is performed.

	FREE TIER
Recall	98/100
Precision	98/98

TABLE III
SYSTEM PERFORMANCE (NATIVE RESOLUTION)

Median precision	98.9%
Median recall	93.9%
Median execution time	2 minutes 29 seconds

TABLE IV
SYSTEM PERFORMANCE (RESCALED PHOTOS)

	960 px	1600 px	2048 px	3072 px
Median precision	95.2%	98.8%	98.9%	98.9%
Median recall	13.5%	67.9%	84.2%	91.4%

with known artifacts that affect the system recall [11]; This phoenomenon can be easily noticed from Fig. 4.

- The runtime is mostly affected by the features of robustness to crops and rotations and to bottlenecks due to Aerospike architecture. If desired, a system that avoids checking automatic crops would be 2x faster for 3:2 cameras and 4x faster for 4:3 or 16:9 cameras, leading to a throughput of the order of more than 700,000 pictures/second.

Figure 6 shows the system precision and recall as a function of the threshold for the second stage, so that it can be used to tweak the precision/recall trade-off. It can be noticed that the chosen value for the threshold ensures a good trade-off as it is close to the saturation point the recall curve and it is before the dropoff in the precision curve.

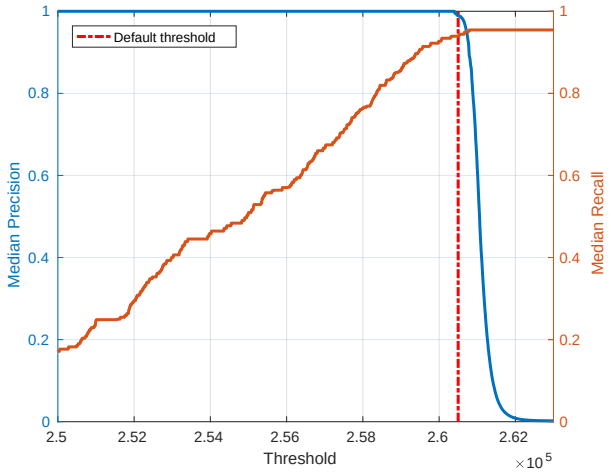


Fig. 6. ToothPic system median precision and recall vs. threshold

Finally, we also tested the system performance in presence of rescaled photos. We rescaled all the Dresden natural photos at four different resolutions corresponding to horizontal resolutions of 960, 1600, 2048, 3072 pixels. Table IV reports the median precision and recall broken down for each resolution. It can be noticed that the system performance degrades smoothly with lowering the resolution due to the lower quality of the fingerprints caused by the rescaling and double JPEG compression.

IV. Conclusions

While ToothPic is the first system to perform camera-based image retrieval on large scales, many problems remain open for future work. In particular, unsupervised image clustering by camera on large scales and resilient to image transformations remains an issue. Camera identification from video sequences is also interesting but challenging due to the lower quality of video frames.

References

[1] J. Fridrich, "Digital image forensics," *Signal Processing Magazine, IEEE*, vol. 26, no. 2, pp. 26–37, 2009.

[2] D. Valsesia, G. Coluccia, T. Bianchi, and E. Magli, "Compressed fingerprint matching and camera identification via random projections," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1472–1485, July 2015.

[3] ——, "Large-scale image retrieval based on compressed camera identification," *IEEE Transactions on Multimedia*, vol. 17, no. 9, pp. 1439–1449, Sept 2015.

[4] M. Chen, J. Fridrich, M. Goljan, and J. Lukas, "Determining image origin and integrity using sensor noise," *IEEE Transactions on Information Forensics and Security*, vol. 3, no. 1, pp. 74–90, March 2008.

[5] D. Valsesia and E. Magli, "Binary adaptive embeddings from order statistics of random projections," *IEEE Signal Processing Letters*, vol. 24, no. 1, pp. 111–115, Jan 2017.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[7] V. Srinivasan, B. Bulkowski, W.-L. Chu, S. Sayyaparaju, A. Gooding, R. Iyer, A. Shinde, and T. Lopatic, "Aerospike: Architecture of a real-time operational dbms," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1389–1400, Sep. 2016. [Online]. Available: http://dx.doi.org/10.14778/3007263.3007276

[8] J. Pokorny, "Nosql databases: A step to database scalability in web environment," in *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services*, ser. iiWAS '11. New York, NY, USA: ACM, 2011, pp. 278–283. [Online]. Available: http://doi.acm.org/10.1145/2095536.2095583

[9] "The Programming Language LUA," https://www.lua.org/, accessed: 2017-11-30.

[10] T. Gloe and R. Böhme, "The Dresden Image Database for benchmarking digital image forensics," *Journal of Digital Forensic Practice*, vol. 3, no. 2-4, pp. 150–159, 2010. [Online]. Available: http://http://forensics.inf.tu-dresden.de/ddimgdb/

[11] T. Gloe, S. Pfennig, and M. Kirchner, "Unexpected Artefacts in PRNU-based Camera Identification: A 'Dresden Image Database' Case-study," in *Proceedings of the on Multimedia and Security*. ACM, 2012, pp. 109–114.

Diego Valsesia is a Postdoctoral Associate at the Department of Electronics and Telecommunications (DET), Politecnico di Torino, Torino, Italy. His main research interests include compression of remote sensing images, compressed sensing, and deep learning.

Giulio Coluccia is a Postdoctoral Associate at the Department of Electronics and Telecommunications (DET), Politecnico di Torino, Torino, Italy. His research is focused on Compressed Sensing, with particular interest in its application to Image Processing and Forensics, Multidimensional Signals and to Distributed Source Coding and Wireless Sensor Networks.

Tiziano Bianchi is an Assistant Professor with the Department of Electronics and Telecommunications, Politecnico di Torino, Torino, Italy. His research interests include multimedia security technologies, signal processing in the encrypted domain, and security aspects of compressed sensing. He has published more than 100 papers on international journals and conference proceedings.

Enrico Magli is a Full Professor with Politecnico di Torino, Torino, Italy, a Fellow of the IEEE and has been an IEEE Distinguished Lecturer from 2015 to 2016. He is an Associate Editor of the IEEE Transactions on Multimedia and the EURASIP Journal on Image and Video Processing. His research interests include compressive sensing, image and video coding, and vision.